(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 800 142 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
24.11.1999 Bulletin 1999/47

(51) Int Cl.[6]: **G06F 9/44**, G06F 17/30

(21) Application number: **97200813.0**

(22) Date of filing: **20.03.1997**

(54) Method for path name format conversion

Verfahren zur Formatkonvertierung bei Pfadnamen

Procédé de conversion de format des noms d'accès

(84) Designated Contracting States:
DE FR GB IT SE

(30) Priority: **01.04.1996 US 626716**

(43) Date of publication of application:
08.10.1997 Bulletin 1997/41

(73) Proprietor: **SUN MICROSYSTEMS, INC.**
Palo Alto, California 94303 (US)

(72) Inventors:
- **Harper, James M.**
  Colorado Springs, CO 80907 (US)
- **Berliner, Brian**
  Colorado Springs, CO 80919 (US)

(74) Representative: **Hanna, Peter William Derek**
Tomkins & Co.,
5 Dartmouth Road
Dublin 6 (IE)

(56) References cited:
EP-A- 0 588 488

- **ADA LETTERS, vol. 10, no. 1, January 1990 - February 1990, NEW YORK, NY, US, pages 111-117, XP000430618 Y. E. GAIL WANG: "UNIVERSAL_FILE_NAMES FOR Ada"**
- **1990 IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS - CONFERENCE PROCEEDINGS, 4 - 7 November 1990, LOS ANGELES, CA, US, pages 499-504, XP000215394 M. DAVIS ET AL: "Unicode"**
- **IBM TECHNICAL DISCLOSURE BULLETIN, vol. 32, no. 10a, March 1990, ARMONK, NY, US, pages 456-462, XP000083401 "FILE NAME MAPPING IN A HETEROGENEOUS DISTRIBUTED ENVIRONMENT"**

EP 0 800 142 B1

Printed by Jouve, 75001 PARIS (FR)

Description

FIELD OF THE INVENTION:

[0001] This invention relates to a method for converting a string of formatted computer readable characters to a new string having a new format.

DESCRIPTION OF RELATED ART:

[0002] The move in the microcomputer industry from the Microsoft@ Windows™ version 3.X graphic environment, which runs under the 8/16-bit MS-DOS operating system, to the 32-bit Microsoft Windows 95 operating system has created a number of problems. Although Windows 95 has been designed so that it will run most software written for Windows ver. 3.X, programs which are written to take full advantage of the expanded capabilities of Windows 95 are often incompatible with Windows ver. 3.X.

[0003] One of the incompatibilities between the old and new versions of Windows relates to path names. A path name is a unique identifier assigned to each file in a distributed data processing system. A path name, as it appears to the end user, has a similar format under both Windows ver. 3.X and Windows 95. The path name for any file always begins with a host identifier and ends with a file name. For example, "C:\WINDOWS\SYS\FILE.EXE" and "\\HOST\J\WINDOWS\SYS\FILE.EXE" are path names representative of those used with Windows ver 3.1. The first example does not follow the *U*niform *N*aming *C*onvention (*UNC*); the second example does. Directories, subdirectories and file names are limited to eight characters plus a three character extension. Generally, however, it is customary to use three character extensions for only file names. The UNC path name contains a host designator "host" and a shared designator "J". It will be noted that the UNC path name begins with double backslash characters, and that no colon character is used in the UNC name.

[0004] Windows ver. 3.X uses *A*merican *S*tandard *C*ode for *I*nformation *I*nterchange (*ASCII* for short). As each ASCII character is represented by a single byte (eight bits), 256 characters are the maximum that may be represented by the ASCII character set. Given the substantial decreases in the cost of both semiconductor and rotating magnetic storage memory, it is not surprising that ASCII code is being supplanted by a new, less-efficient but more capable code based on two-byte characters. The new code, called unicode, can distinguish between 65,536 characters, a quantity sufficient to encompass all foreign alphabets and the Chinese/Kanji characters.

[0005] A path name under Windows ver. 3.X is stored as it appears on a computer screen. That is to say that, in the case of a UNC path name, it is stored as a string of ASCII characters which begins with the double backslash ("\\") and ends with the file name. The elements of the path name (e.g. host, shared, directory, subdirectories, and file name) are separated from one another by a single backslash ("\"). The string is terminated by a null byte (i.e. one which is set to 00HEX). A non-UNC path name is stored beginning with a letter for the drive designator and ends with the file name.

[0006] A path name under Windows 95, on the other hand, is stored as a specially formatted unicode string known as parsed path name structure. Such a structure omits both the host name and the shared name in the case of a UNC name, or the drive designator in the case of a non-UNC path name, and has the following format: The first unicode value, or byte pair, specifies the total length of the parsed path name structure in bytes; the second unicode value indicates the length, in bytes, of the prefix (in this case, the prefix begins with the first unicode value of the string, and includes the directory and all subdirectories up to, but not including, the file name); the third byte pair indicates the length, in bytes, of the first path element in the prefix after the shared drive specifier or initial drive specifier (in the case of a non-UNC name), including the backslash character with which it begins; the third byte pair is followed by an array of unicode values corresponding to the characters in the name of the first path element; the first path element name array is followed by a subsequent byte pair which indicates the length, in bytes, of a second path element (if any); other subsequent byte length indicators and unicode arrays may follow, depending on the number of subdirectories which precede the file name; the final path element is the file name, and it, like other path element, is preceded by a byte length indicator.

[0007] The need for the present invention arose during the development by Sun Microsystems, Inc. of a caching program module which would operate under both Windows 95 and Windows ver. 3.1 in a networked environment which may include modem connections. The module was initially written to operate with Windows 95. As such, it expected inputs, such as parsed path name structures, which are peculiar to Windows 95. If the program were to be run in a Windows ver. 3.X environment, the zero-terminated path names characteristic of Windows ver. 3.X would require conversion to the parsed path name structurcs.

[0008] EP-A-0,588,488 describes a method for accessing the same computer file using different file name formats, of different operating systems. A directory specific request is received from a user. This directory specific request may contain no prefix or suffix, or this directory specific request may contain one of number of operating system specific

prefixes/suffixes. Any prefix or suffix that is contained in the directory specific request is identified, for example DOS, MACINTOSH, OTHER. When no prefix/suffix is used in the request, the specified directory is searched, and it is determined if the requested item is found in the specified directory. When use of a prefix/suffix is found in the directory specific request, the found prefix/suffix is stripped, and a lookup strategy is used that is related to the found prefix/suffix. It is then determined if the requested item was found in the specified directory using the prefix/suffix related lookup strategy of one of the previous steps. Thus, four search strategies are provided; i.e., (1) when no prefix/suffix is used, (2) when the DOS prefix/suffix is used, (3) when the MACINTOSH prefix/suffix is used, and (4) when the OTHER prefix/suffix is used.

[0009] IBM Technical Disclosure Bulletin, Vol.32, no. 10a, March 1990, pp. 456 - 462 "File Mapping in a heterogeneous distributed environment" discloses a method for converting path names between different formats. A method is described that handles the mapping of FILE IDs from one operating system to a second operating system. In order to accomplish this, a PROFILE must be created before mapping use. This PROFILE contains a set of RULES that determine how a given FILE ID is mapped from a first operating system to a second operating system pair, and the details of the RULES must differ depending upon the specific operating systems involved in the operating system pair since a logical connection is established between two different operating systems for each different RULE.

[0010] M. Davis et al in "Unicode"; 1990 IEEE International Conference on Systems, Man and Cybernetics, Conference Proceedings, 4 - 7 November 1990, Los Angeles, CA, USA, pages 499 - 504, discusses the fundamental design principles for encoding characters in computers using Unicode.

SUMMARY OF THE INVENTION

[0011] The present invention provides a method for converting an ASCII path name to a parsed path name structure in accordance with the appended claims.

[0012] In accordance with this invention, the above problem has been solved by the development of an efficient method implemented in conjunction with a computing system for converting a zero-terminated ASCII path name to a parsed path name structure.

[0013] The above computer implemented steps in another implementation of the invention are provided as an article of manufacture, i.e., a computer storage medium containing a computer program of instructions for performing the above described steps.

[0014] The great advantage and utility of the present invention is the efficient conversion of ASCII path names to parsed path name structures. The invention permits program modules designed to run under operating systems that provide parsed path name structure inputs to run under other operating systems which provide only ASCII path name inputs.

[0015] The foregoing and other features, utilities and advantages of the invention will be apparent from the following more particular description of a preferred embodiment of the invention as illustrated in the accompanying object code listing and in the drawings.

Brief Description of Drawings

[0016]

Fig. 1 illustrates a representational computing system and operating environment for performing the computer implemented steps of the method in accordance with the invention; and

Fig. 2 is a flow chart depicting buffer contents at various stages during the conversion of a non-UNC ASCII path name to a parsed path name structure; and

Fig. 3 is a flow chart depicting buffer contents at various stages during the conversion of a UNC ASCII path name to a parsed path name structure.

Detailed Description of Preferred Embodiments

[0017] The embodiments of the invention described herein may be implemented as logical operations in a distributed processing system having client and server computing systems. The logical operations of the present invention are implemented (1) as a sequence of computer implemented steps running on the computing system and (2) as interconnected machine modules within the computing system. The implementation is a matter of choice that is dependent on the performance requirements of the computing system implementing the invention. Accordingly, the logical operations making up the embodiments of the invention described herein are referred to variously as operations, steps or modules.

[0018] The operating environment in which the present invention is used encompasses the general distributed computing system, wherein general purpose computers, workstations, or personal computers are connected via commu-

nication links of various types, in a client-server arrangement, wherein programs and data, many in the form of objects, are made available by various members of the system. Some of the elements of a general purpose workstation computer are shown in Figure 1, wherein a processor 1 is shown, the processor having an input/output (I/O) section, a central processing unit (CPU) 3 and a memory section 4. The I/O section 2 is connected to a keyboard 5, a display unit 6, a disk storage unit 9 and a CD-ROM drive unit 7. The CD-ROM unit 7 can read a CD-ROM medium 8 which typically contains programs 10 and data. The computer program products containing mechanisms to effectuate the apparatus and methods of the present invention may reside in the memory section 4, or on a disk storage unit 9, or on the CD-ROM 8 of such a system. Examples of such systems include SPARC® systems offered by Sun MicroSystems, Inc., personal computers offered by IBM Corporation and by other manufacturers of IBM-compatible personal computers, and systems running the UNIX® operating system.

[0019] As a starting point for describing the invention, an object code listing of a computer program which implements the method is provided. The program, written in C code, is considered to be a preferred implementation of the method. Although other high-level languages might be used to implement the method, C is considered to be particularly well-suited, as large portions of Windows ver. 3.X and Windows 95 are believed to have been written in C.

Conversion Routine in C

```
typedef ParsedPath *path_t;
typedef unsigned short *string_t;

struct PathElement {
      unsigned short  pe_length;
      unsigned short  pe_unichars[1];
}; /*Path Element */

struct ParsedPath {
      unsigned short  pp_totalLength;
      unsigned short  pp_prefixLength;
      struct PathElement pp_elements[1]
}; /* ParsedPath */

//From sunwifs.c

//NameJam - converts ASCII pathname into unicode/path struct
INT
NameJam(PPCHAR pPathName,
      PCRS pcrs,
      path_t *pppath,
      string_t *puFName,
      string_t *pupath)

      INT Len;
      USHORT chCount;
      string_t UniBuf;
      string_t UniPath;
      string_t pUniEnd;
      string_t pUni;
      string_t uFName;
      PUCHAR pStr;
      PUCHAR PathName = *pPathName;
      UNIT DriveNo;

      *pPathName = (PCHAR)LowBuf;
      PathName = LowBus;

      if (pupath) {
            pUniEnd = UniBus = *pupath;

            //Convert the ASCII pathname to unicode
            for (pStr = PathName; *pStr; )
                  *pUniEnd++ = *pStr++;
            *pUniEnd-- = '\0';
      // Fing last '\' for uFName
      for (uFName = pUniEnd; *uFName != '\\'; uFName--)
            ;
      // And point to the next character
      uFName++;

      *puFName = uFName;
}

// if UNC, only convert after the share name
```

```
if (PathName[0] == '\\' && PathName[1] == '\\') {          //UNC
      // first, advance to beginning of share name
      PathName = strchr(&PathName[2], '\\');
      // then, advance to beginning of pathname after share name
      if (PathName)
            PathName = strchr(PathName+1, '\\');
      if (PathName == NULL) {
            return (-1);
      }
      // move back 2 characters to hold ParsedPath structure
      PathName -= 2;

Len = strlen(PathName);
pUniEnd = UniPath = (string_t) *pppath;

// Convert the ASCII pathname to unicode...
for (pStr = PathName; *pStr; )
      *pUniEnd++ = *pStr++;
*pUniEnd-- = '\0';

// Find last '\' for pp prefixLength
for (pUni = pUniEnd; *pUni != '\\'; pUni--)

UniPath[1] = (pUni - UniPath) *2;        //pp_prefixLength
UniPath[0] = 2 * Len;     //pp_totalLength (w/o trailing null)

/** Now back up, counting characters, replacing the '\' with the
  * count (i.e., setting pe_length) */
for (chCount = 0; pUniEnd >= &UniPath[2]l pUniEnd--)
      if (*pUniEnd == '\\' {
            *pUniEnd-- = ++chCount * 2;      //pe_length
            chCount = 0;
      }
      chCount++;
}
```

[0020] Although it is assumed that those having ordinary skill in the art of computer programming will fully understand the logic and function of the heretofore listed computer program, a general description of the program is provided to assist the reader.

[0021] The program begins with several type definition statements and with the definition of a parsed path structure. The parsed path structure is defined as being comprised of multiple elements, each of which is an array of unicode characters.

[0022] The "NameJam" program converts the ASCII pathname first to an unparsed unicode character string and later converts the unparsed string to a parsed path name structure. The NameJam program begins with the definition of various pointers and the allocation of various stack variables. The parsed path name structure will be assembled in a buffer designated pppath. The ASCII path name is loaded in a buffer designated PathName.

[0023] Prior to converting the ASCII path name to the unparsed unicode character string, the NameJam program must first determine whether or not the ASCII path name follows the Uniform Naming Convention (UNC). An ASCII path name which does not follow the UNC begins with a drive specifier character followed, respectively, by a colon character, a single backslash character, a prefix character string, another single backslash character, and a file name character string. On the other hand, an ASCII path name which follows the UNC begins with a pair of adjacent backslash characters followed, respectively, by a host name character string, a single backslash character, a shared name character string (which may be only a single character), a single backslash character, a prefix character string (which may include several elements separated by backslash characters), another single backslash character, and a file name character string. It should be noted that each backslash character preceding a path element is considered to be a part of that path element. Examples of both a non-UNC ASCII path name and a UNC ASCII path name are, respectively:

C:\WINDOWS\SYS\FILE.EXE (non-UNC)

and

\\DENVER\J\WINDOWS\SYS\FILE.EXE (UNC)

[0024] If the ASCII path name follows the UNC, only the portion of the path name after the shared name and beginning with a backslash character is converted to a unicode character string and two dummy characters are placed at the beginning of the string. Alternatively, the conversion is begun two characters to the left of the fourth backslash character

(i.e., at the backslash character immediately to the left of the character "J" in the UNC example above) and no dummy characters are placed at the beginning of the string. In either case, the result is the same.

[0025] If the ASCII path name does not follow the UNC, it is assumed that the path name begins with a drive specifier, followed by a colon character and a backslash character, respectively. The entire non-UNC path name is converted to a unicode character string.

[0026] A character by character conversion of the ASCII path name to unicode is effected, and the buffer designated pppath is sequentially filled with the unicode characters, thus creating an unparsed unicode character string.

[0027] After all required character codes of the ASCII path name are converted to unicode values, and the unicode values have been written to the pppath buffer, a null unicode character is appended to the end of the unparsed string.

[0028] A first pointer is set on the first unicode character of the unparsed string within buffer pppath. For the non-UNC example listed above, the first character is "C"; for the UNC example listed above, the first character is either the first extra character or the backslash ("\") character immediately preceding the character "J".

[0029] The contents of buffer pppath are then scanned and counted in a direction from beginning to end, beginning with the first character in the unparsed string, and stopping on the appended null character. Let us call the total number of characters counted T+1, as this quantity includes the null character.

[0030] The second pointer is then shifted one character toward the first character so that it points to the last character in the string that is immediately adjacent the appended null character. The count is adjusted during the shift so that the total number of characters in the unparsed string is determined to be the quantity T. T is doubled to reflect the fact that each unicode character value contains two bytes. Thus, for both the UNC and non-UNC examples above, 2T = 46 bytes. The first character position in the pppath buffer (i.e., the "C" for the non-UNC case and the first extra character, or the "\" immediately to the left of the character "J" for the UNC case) replaced with the unicode value for the number 46.

[0031] A scanning operation is then performed with a third pointer, beginning with the unicode character pointed at by the second pointer (i.e., the character before the appended null), and in a direction toward the first pointer, stopping on the first backslash character encountered, and counting the number of characters, F, beginning with the character pointed at by the second pointer and ending with the first backslash character on which the third pointer stopped. For both the non-UNC and the UNC cases, the file name length F in unicode characters is equal 9. Thus the file name has a byte length of 2F, or 18 bytes. The backslash character to which the third pointer still points is replaced with the unicode value for the number 18.

[0032] A number P is determined, which represents the length, in bytes, of the prefix. P is calculated by subtracting 2F from 2T, or F from T and doubling the result. For both the UNC and non-UNC cases shown above, P = 28 bytes. The second unicode character in the buffer (e.g., the colon for the non-UNC case and the second extra character or the character "J" for the UNC case are each replaced with the unicode value for the number 28.

[0033] The scanning and counting process is repeated, each time moving in a direction toward the first pointer to the next backslash character, counting the number of characters in that string element, multiplying the number of characters by 2 to obtain the string element byte length, and replacing the backslash character immediately preceding each element with a unicode character which specifies the length of the element in bytes. For example, the backslash character immediately preceding the element "SYS" for both UNC and non-UNC strings is replaced with the unicode value for the number 8, which is the byte length of that element including the initial backslash, and the backslash character immediately preceding the element "WINDOWS" for both UNC and non-UNC strings is replaced with the unicode value for the number 16.

[0034] At this stage of the process, the unparsed unicode string in buffer pppath has been completely converted to a parsed path name structure.

[0035] The flow chart of Figure 2 depicts the contents of the PathName buffer and the pppath buffer at various stages during the conversion of a non-UNC ASCII path name to a parsed path name structure. The non-UNC ASCII path name is first loaded into the PathName buffer 21. Although the contents of PathName buffer 21 are depicted by characters, each byte-wide memory location within the PathName buffer 21 actually contains the binary ASCII code which represents the character. For example, the character "C" would be represented by the binary code "01100111" which is expressed in hexadecimal notation as "67".

[0036] Still referring to Figure 2, for a non-UNC name, the entire path name is converted character-by-character to two-byte-wide unicode and sequentially loaded into the pppath buffer 22A as an un-parsed unicode character string. A null character, represented by sixteen zeros or "00 00" in hexidecimal code, is appended to the unparsed unicode character string.

[0037] Still referring to Figure 2, the unparsed character string is then converted to a parsed path name structure, which is depicted by pppath buffer 22B (the same buffer as 22A, but with different contents) by converting the code for the first character (the letter "C") to a unicode numerical value which represents the length of the unicode character string, minus the null character, in bytes. Thus, the unicode value for "C" is replaced by the unicode value for 46 (the string length in bytes), which is represented by "00 2E" in hexidecimal notation. The unicode value for the colon character, "00 3A" is replaced by the unicode value for 28 (the length of the prefix in bytes), which is represented by "00

1C" in hexidecimal notation. The prefix, of course, comprises every character of the string from "C" to the third "S". The unicode value for each backslash character ("\") is replaced by the unicode value which corresponds to the length, in bytes, of the immediately following path element. That is to say, the first "00 2F" is replaced by "00 10", which represents the byte length of the unicode element "\WINDOWS"; the second "00 2F" is replaced by "00 08", which represents the byte length of the unicode element "\SYS"; and the third "00 2F" is replaced by "00 12", which represents the byte length of the unicode element "\FILE.EXE".

[0038] Conversion of a UNC ASCII path name to a parsed path name structure proceeds by a slightly different process than that for non-UNC ASCII path names. The flow chart of Figure 3 depicts the contents of the PathName buffer and the pppath buffer at various stages during the conversion of a UNC ASCII path name to a parsed path name structure. The entire UNC ASCII path name is first loaded into the PathName buffer 31. It will be remembered that a UNC ASCII path name begins with two adjacent backslash characters. Thus, when the ASCII path name is scanned, the program identifies such a path name as a UNC path name. Those which begin with a single backslash are assumed to begin with a drive identifier, which is followed by a colon, the prefix and the file name. As in Figure 2, although the contents of PathName buffer 21 are depicted by characters, each byte-wide memory location within the PathName buffer 21 actually contains the binary ASCII code which represents the character.

[0039] Still referring to Figure 3, for a UNC name, the portion of the path name to the left of the second character to the left of the prefix is discarded during the conversion of the ASCII path name character codes to unicode. The pppath buffer 32A contains the balance of the path name string, as converted to unicode. Conversion to the final parsed path name structure is effected in a manner identical to that employed for a non-UNC ASCII path name. The result is shown in pppath buffer 32B (the same buffer as buffer 32A, but with different contents).

[0040] While the invention has been particularly shown and described with reference to a preferred embodiment thereof, it will be understood by those skilled in the art that various other changes in the form and details may be made therein without departing from the spirit and scope of the invention. For example, various techniques and sequences may be employed to count characters, determine element lengths, and substitute element length numbers for backslash characters, blank space characters and other characters at the beginning of the prefix.

Claims

1. A method for automatically converting an ASCII path name to a parsed path name structure (22B, 32B), said ASCII path name having a file name element and a prefix element, said prefix element having at least one subelement, said method comprising the computer implemented steps of:

   (a) defining a parsed path name structure;
   (b) allocating buffers for stack variables;
   (c) defining a set of scanning and counting functions and a set of arithmetic and character replacement functions and creating a plurality of pointers for use with said scanning and counting functions and said arithmetic and character replacement functions;
   (d) determining whether or not Uniform Naming Convention is used for the ASCII path name;
   (e) converting each character of the ASCII path name or a subset thereof to a corresponding unicode character, and sequentially writing each unicode character to a buffer to form an unparsed unicode string (22A, 32A);
   (f) employing the scanning and counting functions, in combination with the arithmetic and character replacement functions to convert a first pair of buffer locations to numerical values which are indicative of the total unicode string length and the prefix element length, respectively, and to replace each unicode character corresponding to a backslash character within said unicode string with a specific numerical value, indicative of a length, in bytes, associated with the path name element which began with the replaced backslash character.

2. The method of Claim 1, in which the step of defining a parsed path name structure comprises defining a string of unicode characters, said string having a first unicode character which is assigned a first numerical value specifying a length, in bytes, of said string, said string further comprising a first array of unicode characters which represent said prefix element, said first array being followed by a second array of unicode characters which represent the file name element, said first unicode character being followed by a second unicode character which is assigned a second numerical value specifying a length, in bytes, of said prefix element plus four bytes which account for said first and second unicode characters, said second unicode character being followed by a third unicode character which is assigned a third numerical value specifying a length, in bytes of a first subelement of said prefix element, any further subelement of said prefix element and said file name element each being preceded by a corresponding unicode character which is assigned a numerical value specifying its length in bytes.

3. The method of Claim 1, in which the step of determining whether or not the ASCII path name follows a Uniform Naming Convention (UNC) comprises identifying:

an ASCII path name which does not follow said UNC beginning with a drive specifier character followed, respectively, by a colon character, a prefix beginning with a first single backslash character, and a file name beginning with a second single backslash character;
an ASCII path name which follows said UNC beginning with a pair of adjacent backslash characters followed, respectively, by a host name, a shared name which begins with a first single backslash character, a prefix which begins with a second single backslash character, and a file name which begins with a third single backslash character.

4. The method of Claim 3,

wherein, for ASCII path names which do not follow said UNC, each character of the ASCII path name is converted to the corresponding unicode character and written to the buffer to form the unparsed unicode string (22A); and
wherein, for ASCII path names which follow said UNC, two extra unicode characters are written at the beginning of the buffer, and each character of a subset of the ASCII pathname, said subset comprising the prefix element and the file name element, is converted to the corresponding unicode character and placed immediately adjacent said two extra characters, and wherein said two extra characters and the characters corresponding to said prefix element and said file name element comprise the unparsed unicode string (32A).

5. The method of Claim 4, wherein conversion of the unparsed unicode string (22A, 32A) to a parsed path name structure (22B, 32B) comprises the steps of:

(a) determining a number T, which represents a total number of unicode characters in said string beginning with a first character of said string and ending with a null unicode character which represents a last character of the unicode string;
(b) assigning to said first unicode character of said unparsed unicode string a numerical value equivalent to 2T, which will specify the length of said parsed path name structure in bytes;
(c) determining a number P, which represents a total number of unicode characters which comprise said prefix element;
(d) assigning to the second unicode character of said unparsed unicode string a numerical value equivalent to 2P, which specifies the length of said prefix element in bytes.

6. The method of Claim 5, which further comprises the step of determining the length of all subelements of which said prefix element is comprised and replacing a backslash character which begins each subelement within said prefix element with a numerical value specifying the length in bytes of that element.

7. The method of Claim 3,

wherein, for ASCII path names which do not follow said UNC, each character of the ASCII path name is converted to the corresponding unicode character and written to the buffer to form said unparsed unicode string (22A); and
wherein, for ASCII path names which follow said UNC, each character of a subset of said ASCII pathname, said subset comprising only two characters preceding the prefix element, the prefix element, and the file name element, is converted to the corresponding unicode character and written to the buffer to form said unparsed unicode string (32A).

8. The method of Claim 7, wherein conversion of the unparsed unicode string (22A, 32A) to a parsed path name structure (22B, 32B) comprises the steps of:

(a) determining a number T, which represents a total number of unicode characters in said string beginning with a first character of said string and ending with a null unicode character which represents a last character of the unicode string;
(b) assigning to said first unicode character of said unparsed unicode string a numerical value equivalent to 2T, which will specify the length of said parsed path name structure in bytes;
(c) determining a number P, which represents a total number of unicode characters which comprise said prefix

element;
(d) assigning to the second unicode character of the unparsed unicode string a numerical value equivalent to 2P, which specifies the length of said prefix element in bytes.

9. The method of Claim 8, wherein the total unicode string length T is determined by the steps of:

(a) pointing to said first unicode character with a first pointer;
(b) scanning said unparsed unicode string by advancing a second pointer in a direction from said first unicode character towards said null character, counting a total number, T+1, of unicode characters, including the null character;
(c) moving said second pointer towards the first pointer by one unicode character, so that said second pointer points to said last unicode character, subtracting "1", to reflect this one character move, from T+1, and doubling the result to yield a value 2T indicative of the number of bytes required to represent T unicode characters.

10. The method of Claim 8, wherein prefix element length 2P, in bytes is determined by the steps of:

(a) scanning with a third pointer beginning at the unicode character pointed to by said second pointer and in a direction towards said first pointer, stopping said third pointer at a unicode character corresponding to a first backslash character encountered. and counting a number of characters, F, including said first backslash character: and
(b) calculating the quantity, 2P, which represents the prefix element length in bytes, by subtracting F from T and doubling the result.

11. A computer program product for automatically converting an ASCII path name to a parsed path name structure (22B, 32B), said ASCII path name having a file name element and a prefix element, said prefix element having at least one subelement, the computer program product comprising software code modules directly loadable into the internal memory of a digital computer for performing in the appropriate sequence all the steps of the method of any one of the preceding claims, when said product is used to control the operation of a computer.

Patentansprüche

1. Verfahren zum automatischen Umwandeln eines ASCII-Pfadnamens in eine (durch einen Parser) zerlegte Pfadnamenstruktur (22B, 32B), wobei der ASCII-Pfadname ein Dateinamenelement und ein Präfixelement hat, wobei das Präfixelement wenigstens ein Unterelement hat, wobei das Verfahren die folgenden computerimplementierten Schritte aufweist:

(a) Definieren einer zerlegten Pfadnamenstruktur;
(b) Zuteilen von Puffern für Stapelvariable;
(c) Definieren eines Satzes von Abtast- und Zählfunktionen und eines Satzes von Arithmetik- und Zeichen-Ersetz-Funktionen und Erzeugen einer Vielzahl von Zeigern zur Verwendung mit den Abtast- und Zählfunktionen und den Arithmetik- und Zeichen-Ersetz-Funktionen;
(d) Bestimmen, ob eine einheitliche Namensgebungskonvention für den ASCII-Pfadnamen verwendet wird oder nicht;
(e) Umwandeln jedes Zeichens des ASCII-Pfadnamens oder einer Untergruppe davon in ein entsprechendes Einheitscodezeichen und sequentielles Schreiben jedes Einheitscodezeichens zu einem Puffer, um eine nicht zerlegte Einheitscodekette (22A, 32A) zu bilden;
(f) Verwenden der Abtast- und Zählfunktionen in Kombination mit den Arithmetik- und Zeichen-Ersetz-Funktionen, um ein erstes Paar von Pufferstellen in numerische Werte umzuwandeln, die jeweils eine gesamte Einheitscodekettenlänge und die Präfixelementlänge anzeigen, und um jedes Einheitscodezeichen entsprechend einem Invers-Schrägstrich-Zeichen innerhalb der Einheitscodekette durch einen spezifischen numerischen Wert zu ersetzen, der eine Länge in Bytes anzeigt, die zum Pfadnamenelement gehört, das mit dem ersetzten Invers-Schrägstrich-Zeichen begann.

2. Verfahren nach Anspruch 1, wobei der Schritt zum Definieren einer zerlegten Pfadnamenstruktur ein Definieren einer Kette von Einheitscodezeichen aufweist, wobei die Kette ein erstes Einheitscodezeichen hat, das einem ersten numerischen Wert zugeordnet ist, der eine Länge der Kette in Bytes spezifiziert, wobei die Kette weiterhin ein erstes Feld von Einheitscodezeichen aufweist, die das Präfixelement darstellen, wobei dem ersten Feld ein

zweites Feld von Einheitscodezeichen folgt, die das Dateinamenelement darstellen, wobei dem ersten Einheitscodezeichen ein zweites Einheitscodezeichen folgt, das einem zweiten numerischen Wert zugeordnet ist, der eine Länge des Präfixelements in Bytes plus vier Bytes spezifiziert, die das erste und das zweite Einheitscodezeichen kontieren, wobei dem zweiten Einheitscodezeichen ein drittes Einheitscodezeichen folgt, das einem dritten numerischen Wert zugeordnet ist, der eine Länge eines ersten Unterelements des Präfixelements in Bytes spezifiziert, wobei jedem weiteren Unterelement des Präfixelements und des Dateinamenelements jeweils ein entsprechendes Einheitscodezeichen vorangeht, das einem numerischen Wert zugeordnet ist, der seine Länge in Bytes spezifiziert.

3. Verfahren nach Anspruch 1, wobei der Schritt zum Bestimmen, ob der ASCII-Pfadname einer einheitlichen Namensgebungskonvention (UNC) folgt oder nicht, ein Identifizieren von folgendem aufweist:

eines ASCII-Pfadnamens, der der UNC nicht folgt, beginnend mit einem Treiberspezifiziererzeichen, dem jeweils ein Doppelpunkt-Zeichen, ein Präfix beginnend mit einem einzelnen Invers-Schrägstrich-Zeichen und ein Dateiname beginnend mit einem weiteren einzelnen Invers-Schrägstrich-Zeichen folgt;
eines ASCII-Pfadnamens, der der UNC folgt, beginnend mit einem Paar benachbarter Invers-Schrägstrich-Zeichen, dem jeweils ein Host-Name, ein gemeinsam genutzter Name, der mit einem einzelnen Invers-Schrägstrich-Zeichen beginnt, ein Präfix, das mit einem weiteren einzelnen Invers-Schrägstrich-Zeichen beginnt, und ein Dateiname, der mit einem weiteren einzelnen Invers-Schrägstrich-Zeichen beginnt, folgt.

4. Verfahren nach Anspruch 3,

wobei für ASCII-Pfadnamen, die der UNC nicht folgen, jedes Zeichen des ASCII-Pfadnamens in ein entsprechendes Einheitscodezeichen umgewandelt und zum Puffer geschrieben wird, um die nicht zerlegte Einheitscodekette (22A) zu bilden; und
wobei für ASCII-Pfadnamen, die der UNC folgen, zwei besondere Einheitscodezeichen an den Anfang des Puffers geschrieben werden, und jedes Zeichen einer Untergruppe des ASCII-Pfadnamens, wobei die Untergruppe das Unterelement und das Dateinamenelement aufweist, in das entsprechende Einheitscodezeichen umgewandelt und direkt benachbart zu den zwei besonderen Zeichen angeordnet wird, und wobei die zwei besonderen Zeichen und die Zeichen, die dem Unterelement und dem Dateinamenelement entsprechen, die nicht zerlegte Einheitscodekette (32A) aufweisen.

5. Verfahren nach Anspruch 4, wobei eine Umwandlung der nicht zerlegten Einheitscodekette (22A, 32A) in eine zerlegte Pfadnamenstruktur (22B, 32B) die folgenden Schritte aufweist:

(a) Bestimmen einer Zahl T, die eine Gesamtanzahl von Einheitscodezeichen in der Kette beginnend mit einem ersten Zeichen der Kette und endend mit einem Einheitscodezeichen von Null, das ein letztes Zeichen der Einheitscodekette darstellt, darstellt;
(b) Zuordnen eines numerischen Werts gleich 2T zum ersten Einheitscodezeichen der nicht zerlegten Einheitscodekette, der die Länge der zerlegten Pfadnamenstruktur in Bytes spezifizieren wird;
(c) Bestimmen einer Zahl P, die eine Gesamtanzahl von Einheitscodezeichen darstellt, die das Präfixelement aufweisen;
(d) Zuordnen eines numerischen Werts gleich 2P zum zweiten Einheitscodezeichen der nicht zerlegten Einheitscodekette, der die Länge des Präfixelements in Bytes spezifiziert.

6. Verfahren nach Anspruch 5, das weiterhin den Schritt zum Bestimmen der Länge aller Unterelemente aufweist, aus welchen das Präfixelement aufgebaut ist, und Ersetzen eines Invers-Schrägstrich-Zeichens, das jedes Unterelement innerhalb des Präfixelements beginnt, durch einen numerischen Wert, der die Länge jenes Elements in Bytes spezifiziert.

7. Verfahren nach Anspruch 3,

wobei für ASCII-Pfadnamen, die der UNC nicht folgen, jedes Zeichen des ASCII-Pfadnamens in das entsprechende Einheitscodezeichen umgewandelt und zum Puffer geschrieben wird, um die nicht zerlegte Einheitscodekette (22A) zu bilden; und
wobei für ASCII-Pfadnamen, die der UNC folgen, jedes Zeichen einer Untergruppe des ASCII-Pfadnamens, wobei die Untergruppe nur zwei Zeichen aufweist, die dem Präfixelement, dem Präfixelement und dem Dateinamenelement vorangehen, in das entsprechende Einheitscodezeichen umgewandelt und zum Puffer geschrieben wird, um die nicht zerlegte Einheitscodekette (32A) zu bilden.

8. Verfahren nach Anspruch 7, wobei eine Umwandlung der nicht zerlegten Einheitscodekette (22A, 32A) in eine zerlegte Pfadnamenstruktur (22B, 32B) die folgenden Schritte aufweist:

(a) Bestimmen einer Zahl T, die eine Gesamtanzahl von Einheitscodezeichen in der Kette beginnend mit einem ersten Zeichen der Kette und endend mit einem Einheitscodezeichen von Null, das ein letztes Zeichen der Einheitscodekette darstellt, darstellt;
(b) Zuordnen eines numerischen Werts gleich 2T zum ersten Einheitscodezeichen der nicht zerlegten Einheitscodekette, der die Länge der zerlegten Pfadnamenstruktur in Bytes spezifizieren wird;
(c) Bestimmen einer Zahl P, die eine Gesamtanzahl von Einheitscodezeichen darstellt, die das Präfixelement aufweisen:
(d) Zuordnen eines numerischen Werts gleich 2P zum zweiten Einheitscodezeichen der nicht zerlegten Einheitscodekette, der die Länge des Präfixelements in Bytes spezifiziert.

9. Verfahren nach Anspruch 8, wobei die gesamte Einheitscodekettenlänge T durch die folgenden Schritte bestimmt wird:

(a) Zeigen zum ersten Einheitscodezeichen mit einem ersten Zeiger;
(b) Abtasten der nicht zerlegten Einheitscodekette durch Vorrücken eines zweiten Zeigers in einer Richtung vom ersten Einheitscodezeichen zum Zeichen von Null, Zählen einer Gesamtanzahl T+1 von Einheitscodezeichen einschließlich des Zeichens von Null;
(c) Bewegen des zweiten Zeigers in Richtung zum ersten Zeiger um ein Einheitscodezeichen, so daß der zweite Zeiger zum letzten Einheitscodezeichen zeigt, Subtrahieren von "1" von T+1, um diese Bewegung um ein Zeichen zu berücksichtigen, und Verdoppeln des Ergebnisses, damit sich ein Wert von 2T ergibt, der die Anzahl von Bytes anzeigt, die zum Darstellen von T Einheitscodezeichen erforderlich sind.

10. Verfahren nach Anspruch 8, wobei eine Präfixelementenlänge 2P in Bytes durch die folgenden Schritte bestimmt wird:

(a) Abtasten mit einem dritten Zeiger beginnend bei dem Einheitscodezeichen, auf das durch den zweiten Zeiger gezeigt wird, und in einer Richtung zum ersten Zeiger, Anhalten des dritten Zeigers bei einem Einheitscodezeichen entsprechend einem ersten angetroffenen Invers-Schrägstrich-Zeichen und Zählen einer Anzahl von Zeichen F einschließlich des ersten Invers-Schrägstrich-Zeichens; und
(b) Berechnen der Quantität 2P, die die Präfixelementenlänge in Bytes darstellt, durch Subtrahieren von F von T und Verdoppeln des Ergebnisses.

11. Computerprogrammprodukt zum automatischen Umwandeln eines ASCII-Pfadnamens in eine (durch einen Parser) zerlegte Pfadnamenstruktur (22B, 32B), wobei der ASCII-Pfadname ein Dateinamenelement und ein Präfixelement hat, wobei das Präfixelement wenigstens ein Unterelement hat, das direkt in den internen Speicher eines digitalen Computers ladbar ist, der Software-Codeteile zum Durchführen der Schritte eines beliebigen der vorangehenden Ansprüche aufweist, wenn das Produkt auf einem Computer läuft.

Revendications

1. Procédé pour convertir automatiquement un nom de chemin exprimé en ASCII en une structure de nom de chemin analysée (22B, 32B), ledit nom de chemin exprimé en ASCII ayant un élément de nom de fichier et un élément de préfixe, ledit élément de préfixe ayant au moins un sous-élément, ledit procédé comportant les étapes implémentées par ordinateur consistant à :

(a) définir une structure de nom de chemin analysée,
(b) allouer des tampons à des variables de pile,
(c) définir un ensemble de fonctions de balayage et de comptage et un ensemble de fonctions arithmétiques et de remplacement de caractères et créer une pluralité de pointeurs destinés à être utilisés avec lesdites fonctions de balayage et de comptage et lesdites fonctions arithmétiques et de remplacement de caractères,
(d) déterminer si une Convention d'Appellation Uniforme est utilisée ou non pour le nom de chemin exprimé en ASCII,
(e) convertir chaque caractère du nom de chemin exprimé en ASCII ou d'un sous-ensemble de celui-ci en un caractère unicode correspondant, et écrire séquentiellement chaque caractère unicode dans un tampon pour

former une chaîne de caractères unicode non-analysée (22A, 32A),
(f) utiliser les fonctions de balayage et de comptage, en combinaison avec les fonctions arithmétiques et de remplacement de caractères pour convertir une première paire d'emplacements tampon en valeurs numériques qui sont représentatives de la longueur totale de chaîne de caractères unicode et de la longueur d'élément de préfixe, respectivement, et pour remplacer chaque caractère unicode correspondant à un caractère "barre oblique inverse" dans ladite chaîne de caractères unicode par une valeur numérique spécifique, représentative d'une longueur, en octets, associée à l'élément de nom de chemin qui commence par le caractère "barre oblique inverse" remplacé.

2. Procédé selon la revendication 1, dans lequel l'étape consistant à définir une structure de nom de chemin analysée comporte la définition d'une chaîne de caractères unicode, ladite chaîne ayant un premier caractère unicode qui est affecté à une première valeur numérique spécifiant une longueur, en octets, de ladite chaîne, ladite chaîne comportant de plus une première série de caractères unicode qui représente ledit élément de préfixe, ladite première série étant suivie d'une seconde série de caractères unicode qui représente l'élément de nom de fichier, ledit premier caractère unicode étant suivi d'un deuxième caractère unicode qui est affecté à une deuxième valeur numérique spécifiant une longueur, en octets, dudit élément de préfixe complété des quatre octets qui correspondent auxdits premier et deuxième caractères unicode, ledit deuxième caractère unicode étant suivi d'un troisième caractère unicode qui est affecté à une troisième valeur numérique spécifiant une longueur, en octets, d'un premier sous-élément dudit élément de préfixe, tout autre sous-élément dudit élément de préfixe et dudit élément de nom de fichier étant chacun précédé d'un caractère unicode correspondant qui est affecté à une valeur numérique spécifiant sa longueur en octets.

3. Procédé selon la revendication 1, dans lequel l'étape consistant à déterminer si le nom de chemin exprimé en ASCII suit ou non une Convention d'Appellation Uniforme (UNC) comporte l'étape consistant à identifier :

un nom de chemin exprimé en ASCII qui ne suit pas ladite Convention UNC commençant par un caractère spécifiant le répertoire racine suivi, respectivement, du caractère "deux points", d'un préfixe commençant par un caractère "barre oblique inverse" unique, et d'un nom de fichier commençant par un autre caractère "barre oblique inverse" unique,
un nom de chemin exprimé en ASCII qui suit ladite Convention UNC commençant par une paire de caractères "barre oblique inverse" adjacents suivis, respectivement, d'un nom d'hôte, d'un nom partagé qui commence par un caractère "barre oblique inverse" unique, d'un préfixe qui commence par un autre caractère "barre oblique inverse" unique, et d'un nom de fichier qui commence par encore un autre caractère "barre oblique inverse" unique.

4. Procédé selon la revendication 3,

dans lequel, pour des noms de chemin exprimés en ASCII qui ne suivent pas ladite Convention UNC, chaque caractère du nom de chemin exprimé en ASCII est converti en un caractère unicode correspondant et écrit dans le tampon pour former la chaîne de caractères unicode non-analysée (22A), et
dans lequel, pour des noms de chemin exprimés en ASCII qui suivent ladite Convention UNC, deux caractères unicode supplémentaires sont écrits tout au début du tampon, et chaque caractère d'un sous-ensemble du nom de chemin exprimé en ASCII, ledit sous-ensemble comportant le sous-élément et l'élément de nom de fichier, est converti en caractère unicode correspondant et placé immédiatement adjacent auxdits deux caractères supplémentaires, et lesdits deux caractères supplémentaires et les caractères correspondant audit sous-élément et audit élément de nom de fichier comportant la chaîne de caractères unicode non-analysée (32A).

5. Procédé selon la revendication 4, dans lequel la conversion de la chaîne de caractères unicode non-analysée (22A, 32A) en une structure de nom de chemin analysée (22B, 32B) comporte les étapes consistant à :

(a) déterminer un nombre T, qui représente un nombre total de caractères unicode dans ladite chaîne en commençant par un premier caractère de ladite chaîne et en terminant par un caractère unicode nul qui représente un dernier caractère de la chaîne de caractères unicode,
(b) affecter ledit premier caractère unicode de ladite chaîne de caractères unicode non-analysée à une valeur numérique équivalente à 2T, qui va spécifier la longueur de ladite structure de nom de chemin analysée en octets,
(c) déterminer un nombre P, qui représente le nombre total de caractères unicode qui constituent ledit élément

de préfixe,
(d) affecter ledit deuxième caractère unicode de ladite chaîne de caractères unicode non-analysée à une valeur numérique équivalente à 2P, qui spécifie la longueur dudit élément de préfixe en octets.

6. Procédé selon la revendication 5, qui comporte de plus l'étape consistant à déterminer la longueur de tous les sous-éléments qui constituent ledit élément de préfixe et à remplacer un caractère "barre oblique inverse" qui commence chaque sous-élément dans ledit élément de préfixe par une valeur numérique spécifiant la longueur en octets de cet élément.

7. Procédé selon la revendicàtion 3,

dans lequel, pour des noms de chemin exprimés en ASCII qui ne suivent pas ladite Convention UNC, chaque caractère du nom de chemin exprimé en ASCII est converti en caractère unicode correspondant et est écrit dans le tampon pour former ladite chaîne de caractères unicode non-analysée (22A), et
dans lequel, pour des noms de chemin exprimés en ASCII qui suivent ladite Convention UNC, chaque caractère d'un sous-ensemble dudit nom de chemin exprimé en ASCII, ledit sous-ensemble comportant uniquement deux caractères précédant l'élément de préfixe, de l'élément de préfixe et de l'élément de nom de fichier, est converti en caractère unicode correspondant et écrit dans le tampon pour former ladite chaîne de caractères unicode non-analysée (32A).

8. Procédé selon la revendication 7, dans lequel la conversion de la chaîne de caractères unicode non-analysée (22A, 32A) en une structure de nom de chemin analysée (22B, 32B) comporte les étapes consistant à :

(a) déterminer un nombre T, qui représente le nombre total de caractères unicode dans ladite chaîne en commençant par un premier caractère de ladite chaîne et en terminant par un caractère unicode nul qui représente un dernier caractère de la chaîne de caractères unicode,
(b) affecter ledit premier caractère unicode de ladite chaîne de caractères unicode non-analysée à une valeur numérique équivalente à 2T, qui va spécifier la longueur de ladite structure de nom de chemin analysée en octets,
(c) déterminer un nombre P, qui représente le nombre total de caractères unicode qui constituent ledit élément de préfixe,
(d) affecter ledit deuxième caractère unicode de la chaîne de caractères unicode non-analysée à une valeur numérique équivalente à 2P, qui spécifie la longueur dudit élément de préfixe en octets.

9. Procédé selon la revendication 8, dans lequel la longueur totale de chaîne de caractères unicode T est déterminée par les étapes consistant à :

(a) pointer sur ledit premier caractère unicode à l'aide d'un premier pointeur,
(b) balayer ladite chaîne de caractères unicode non-analysée en faisant avancer un deuxième pointeur dans une direction qui va dudit premier caractère unicode audit caractère nul, compter un nombre total, T + 1, de caractères unicode, caractère nul inclus,
(c) déplacer ledit deuxième pointeur vers le premier pointeur de un caractère unicode, de sorte que ledit deuxième pointeur pointe sur ledit dernier caractère unicode, soustraire la valeur "1", pour refléter ce déplacement de un caractère, de T + 1, et multiplier par deux le résultat pour produire une valeur 2T représentative du nombre d'octets nécessaire pour représenter les T caractères unicode.

10. Procédé selon la revendication 8, dans lequel la longueur d'élément de préfixe 2P, en octets, est déterminée par les étapes consistant à :

(a) balayer à l'aide d'un troisième pointeur en commençant au niveau du caractère unicode pointé par ledit deuxième pointeur et dans une direction allant vers ledit premier pointeur, stopper ledit troisième pointeur au niveau d'un caractère unicode correspondant au premier caractère "barre oblique inverse" rencontré, et compter le nombre de caractères, F, ledit premier caractère "barre oblique inverse" inclus, et
(b) calculer la quantité, 2P, qui représente la longueur d'élément de préfixe en octets, en soustrayant F de T et en multipliant par deux le résultat.

11. Produit programme informatique pour convertir automatiquement un nom de chemin exprimé en ASCII en une structure de nom de chemin analysée (22B, 32B), ledit nom de chemin exprimé en ASCII ayant un élément de

nom de fichier et un élément de préfixe, ledit élément de préfixe ayant au moins un sous-élément pouvant être chargé directement dans la mémoire interne d'un calculateur numérique, comportant des parties de code logiciel pour effectuer les étapes de l'une quelconque des revendications précédentes, ledit produit étant exécuté sur un ordinateur.

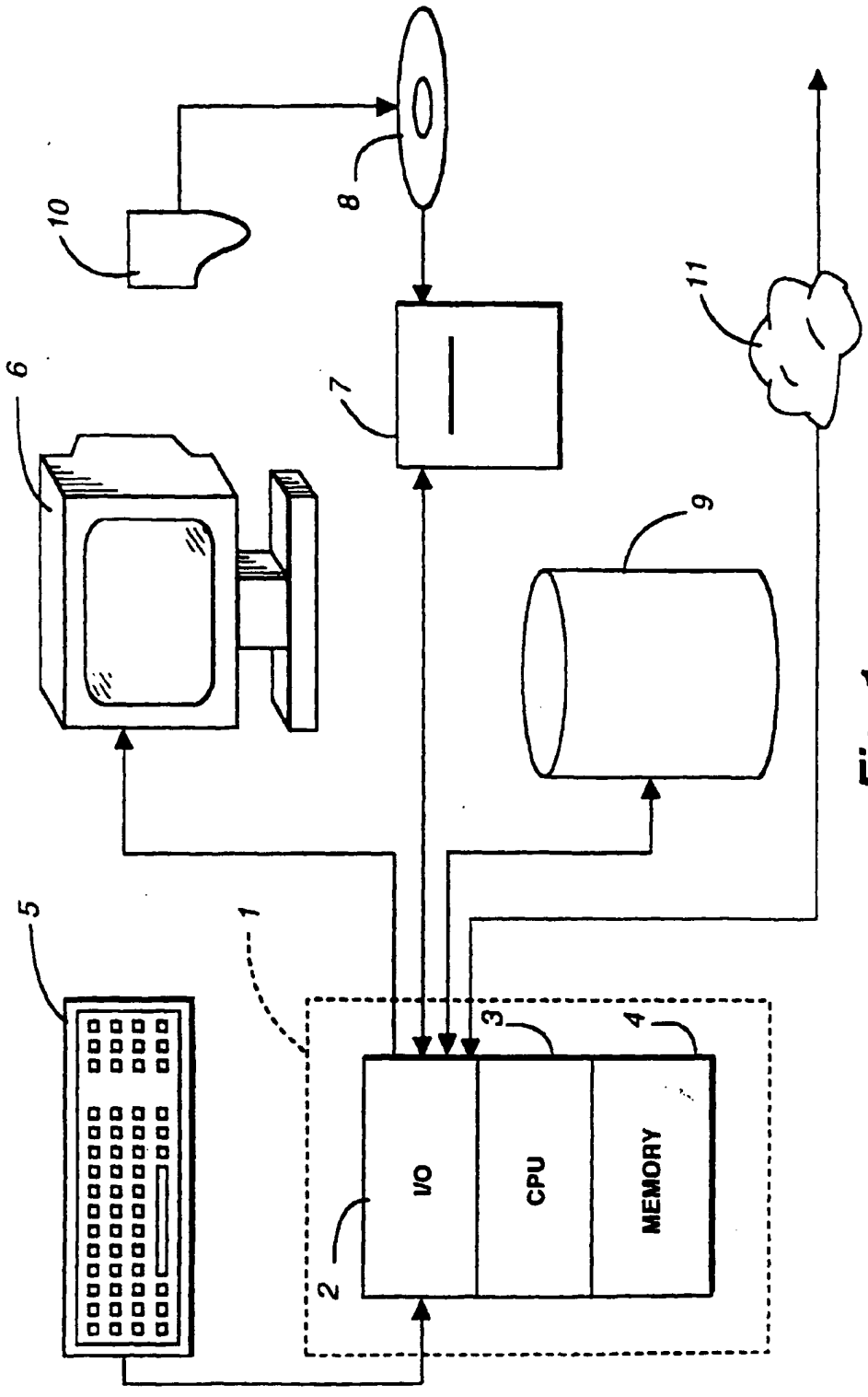

Fig. 1

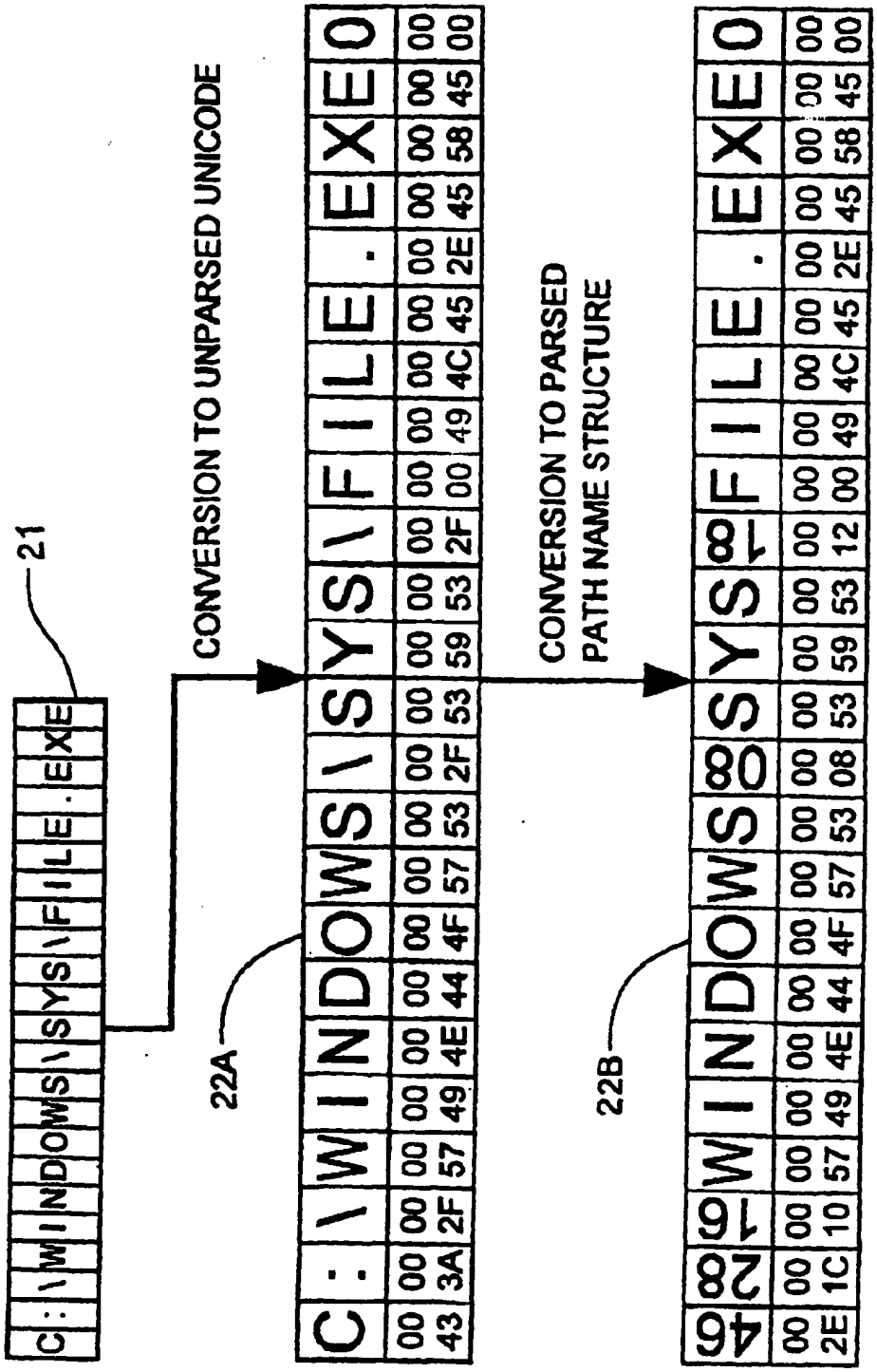
C:\WINDOWS\SYS\FILE.EXE
21
CONVERSION TO UNPARSED UNICODE
22A
CONVERSION TO PARSED
PATH NAME STRUCTURE
22B

Fig. 2

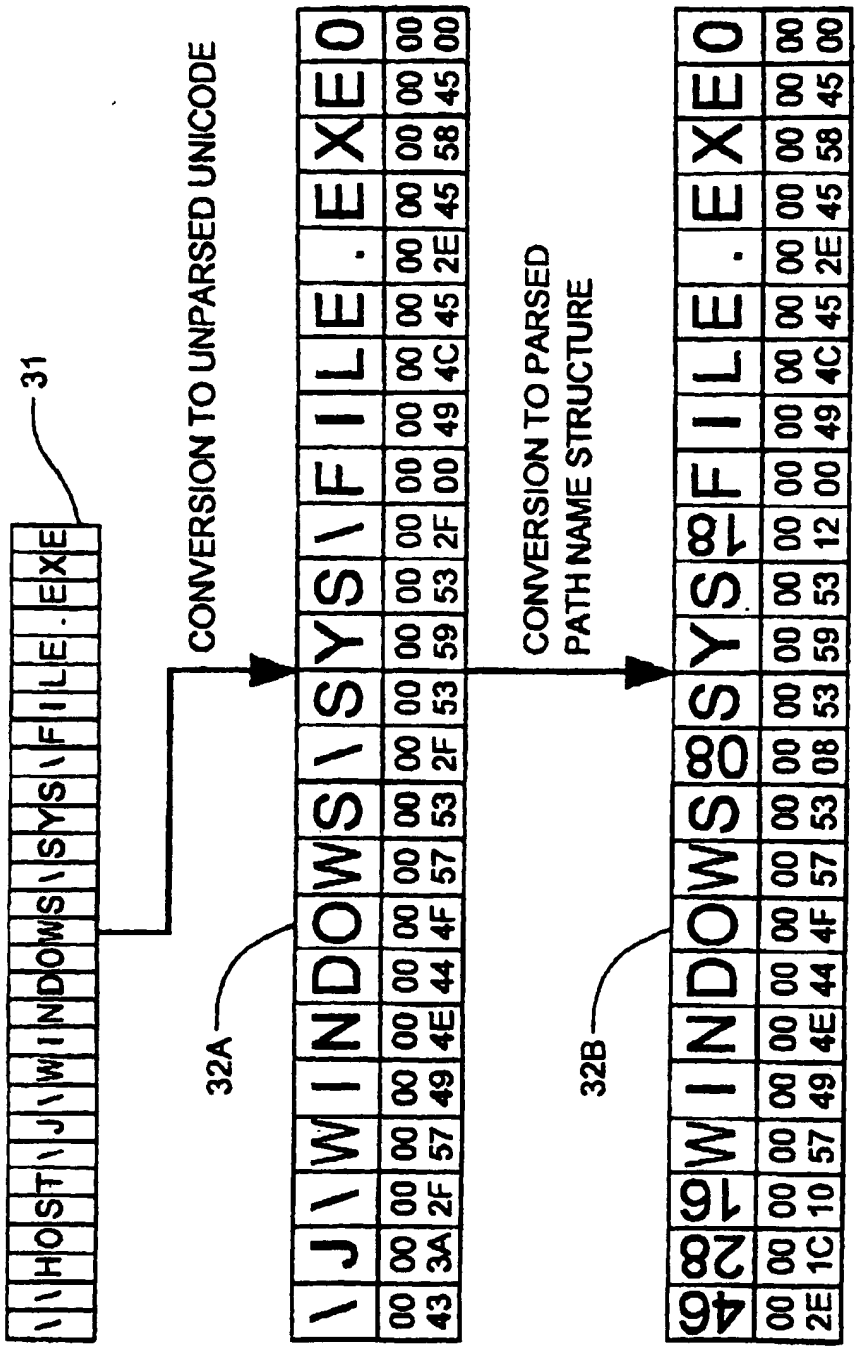
\\HOST\J\WINDOWS\SYS\FILE.EXE
31
CONVERSION TO UNPARSED UNICODE
\J\WINDOWS\SYS\FILE.EXE
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 43 3A 2F 57 49 4E 44 4F 57 53 2F 53 59 53 2F 00 49 4C 45 2E 45 58 45 00 00
32A
CONVERSION TO PARSED
PATH NAME STRUCTURE
46 82 96 WINDOWS 8~1 SYS 8~1 FILE.EXE 0
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 2E 1C 10 57 49 4E 44 4F 57 53 08 53 59 53 12 00 49 4C 45 2E 45 58 45 00 00
32B

Fig. 3